A3: Make Your Own Machine Translation Language

Get Language Pair

```
!pip install sacrebleu
```

```
zsh:1: command not found: pip
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import random
import math
import time
import numpy as np
import pandas as pd
import re
import os
from datasets import load_dataset
import matplotlib.pyplot as plt
import seaborn as sns
from torch.utils.data import DataLoader, Dataset
from collections import Counter
import unicodedata
from sacrebleu.metrics import BLEU
from tqdm.auto import tqdm
```

```
/Users/santhosh/Documents/DSAI/Semester 2/NLP/A3
Assignment/venv/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and
ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Set random seed for reproducibility
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

```
Using device: cpu
```

```
# Constants for special tokens
UNK_IDX, PAD_IDX, SOS_IDX, EOS_IDX = 0, 1, 2, 3
special_symbols = ['<unk>', '<pad>', '<sos>', '<eos>']

# Language constants
SRC_LANGUAGE = 'en'
TRG_LANGUAGE = 'ta'
```

Load Data

Dataset Source & Credits

Dataset Name: OPUS-100 (English-Tamil)
Source: Hugging Face Datasets - Helsinki-NLP/opus-100
Original Repository: OPUS - Parallel Corpus

Citation:

> Zhang, B., Williams, P., Titov, I., & Sennrich, R. (2020). Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 1628–1639. https://arxiv.org/abs/2004.11867

Description: OPUS-100 is an English-centric multilingual corpus covering 100 languages. It was constructed by sampling up to 1 million sentence pairs per language pair from the OPUS collection. The English-Tamil (`en-ta`) subset contains approximately 227k parallel sentences used for training, validation, and testing in this assignment.

```
# Load the English-Tamil dataset
print("Loading dataset...")
dataset = load_dataset("opus100", "en-ta")
print(f"Dataset loaded successfully!")
print(f"Train size: {len(dataset['train'])}")
print(f"Validation size: {len(dataset['validation'])}")
print(f"Test size: {len(dataset['test'])}")

# Print some examples
print("\nExample translations from dataset:")
for i in range(3):
    example = dataset['train'][i]
    print(f"\nExample {i+1}:")
    print(f"English: {example['translation']['en']}")
    print(f"Tamil: {example['translation']['ta']}")
```

```
Loading dataset...
Dataset loaded successfully!
Train size: 227014
Validation size: 2000
Test size: 2000
```

```
Example translations from dataset:

Example 1:
English: The likeness of those who disbelieve in their Lord: their
works are like ashes, in a fierce wind, on a stormy day. They have no
control over anything they have earned. That is the utmost
misguidance.
Tamil: எவர்கள் தங்களுடைய இறைவனை நிராகரிக்கிறார்களோ,
அவர்களுக்கு உதாரணமாவது அவர்களுடைய செயல்கள் சாம்பல்
போன்றவை புயல் காற்று கடினமாக வீசம் நாளில் அச்சாம்பலைக் காற்று
அடித்துக் கொண்டு போய்விட்டது. (அவ்வாறே) தாங்கள் சம்பாதித்த
பொருள்களில் எதன் மீதும் அவர்களுக்கு அதிகாரம் இராது இதுவே வெகு
தூரமான வழிகேடாகும்.

Example 2:
English: This day every soul shall be rewarded for what it has earned;
no injustice (shall be done) this day; surely Allah is quick in
reckoning.
Tamil: அந்நாளில் ஒவ்வோர் ஆத்மாவும், அது சம்பாதித்ததற்குக் கூலி
கொடுக்கப்படும்; அந்நாளில் எந்த அநியாயமும் இல்லை. நிச்சயமாக,
அல்லாஹ் கேள்வி கணக்குக் கேட்பதில் மிகவும் தீவிரமானவன்.

Example 3:
English: svg icon
Tamil: svg யின் சின்னம்.
```

Tokenization

```python
# Tamil text normalization
def normalize_tamil_text(text):
    """
    Normalize Tamil text with specific rules:
    1. NFKC normalization for consistent Unicode representation
    2. Handle Tamil-specific characters and combinations
    3. Standardize numerals and punctuation
    """
    # NFKC normalization
    text = unicodedata.normalize('NFKC', text)

    # Remove extra spaces
    text = ' '.join(text.split())

    # Standardize numerals (optional: convert to Tamil numerals)
    numeral_map = str.maketrans('0123456789', '೦೧೨೩೪೫೬೭೮೯')
    text = text.translate(numeral_map)

    return text
```

```
class CustomTokenizer:
    def __init__(self, texts, max_vocab_size=50000, language='en'):
        print(f"\nInitializing {language} tokenizer...")
        self.max_vocab_size = max_vocab_size
        self.language = language
        self.word2idx = {'<unk>': UNK_IDX, '<pad>': PAD_IDX, '<sos>':
SOS_IDX, '<eos>': EOS_IDX}
        self.idx2word = {v: k for k, v in self.word2idx.items()}
        self.vocab_size = len(special_symbols)

        print(f"Building vocabulary for {language}...")
        # Build vocabulary
        word_freq = Counter()
        for i, text in enumerate(texts):
            if i % 10000 == 0:
                print(f"Processing text {i}/{len(texts)}")

            # Apply language-specific normalization
            if language == 'ta':
                text = normalize_tamil_text(text)
            else:
                text = text.lower()

            words = text.split()
            word_freq.update(words)

        # Add most common words to vocabulary
        for word, freq in word_freq.most_common(max_vocab_size -
len(special_symbols)):
            if word not in self.word2idx:
                self.word2idx[word] = self.vocab_size
                self.idx2word[self.vocab_size] = word
                self.vocab_size += 1

        print(f"Vocabulary size for {language}: {self.vocab_size}")
        print(f"Sample of most frequent words in {language}:")
        for word, freq in list(word_freq.most_common(10)):
            print(f"  {word}: {freq}")

    def encode(self, text):
        if self.language == 'ta':
            text = normalize_tamil_text(text)
        else:
            text = text.lower()
        words = text.split()
        return [SOS_IDX] + [self.word2idx.get(word, UNK_IDX) for word
in words] + [EOS_IDX]

    def decode(self, indices):
```

```
        return ' '.join([self.idx2word.get(idx, '<unk>') for idx in
indices if idx not in [PAD_IDX, SOS_IDX, EOS_IDX]])

print("\nCreating tokenizers...")
src_texts = [example['translation']['en'] for example in
dataset['train']]
trg_texts = [example['translation']['ta'] for example in
dataset['train']]
print(f"Total English texts: {len(src_texts)}")
print(f"Total Tamil texts: {len(trg_texts)}")

src_tokenizer = CustomTokenizer(src_texts, language='en')
trg_tokenizer = CustomTokenizer(trg_texts, language='ta')

# Add tokenization examples
print("\nTokenization examples:")
for i in range(3):
    example = dataset['train'][i]
    en_text = example['translation']['en']
    ta_text = example['translation']['ta']

    en_tokens = src_tokenizer.encode(en_text)
    ta_tokens = trg_tokenizer.encode(ta_text)

    print(f"\nExample {i+1}:")
    print(f"English: {en_text}")
    print(f"Tokenized English: {en_tokens}")
    print(f"Decoded English: {src_tokenizer.decode(en_tokens)}")
    print(f"Tamil: {ta_text}")
    print(f"Tokenized Tamil: {ta_tokens}")
    print(f"Decoded Tamil: {trg_tokenizer.decode(ta_tokens)}")


Creating tokenizers...
Total English texts: 227014
Total Tamil texts: 227014

Initializing en tokenizer...
Building vocabulary for en...
Processing text 0/227014
Processing text 10000/227014
Processing text 20000/227014
Processing text 30000/227014
Processing text 40000/227014
Processing text 50000/227014
Processing text 60000/227014
Processing text 70000/227014
Processing text 80000/227014
Processing text 90000/227014
Processing text 100000/227014
```

```
Processing text 110000/227014
Processing text 120000/227014
Processing text 130000/227014
Processing text 140000/227014
Processing text 150000/227014
Processing text 160000/227014
Processing text 170000/227014
Processing text 180000/227014
Processing text 190000/227014
Processing text 200000/227014
Processing text 210000/227014
Processing text 220000/227014
Vocabulary size for en: 50000
Sample of most frequent words in en:
  the: 139279
  and: 106945
  of: 68671
  to: 65329
  you: 55207
  is: 45289
  a: 42650
  they: 39710
  in: 36398
  for: 29564

Initializing ta tokenizer...
Building vocabulary for ta...
Processing text 0/227014
Processing text 10000/227014
Processing text 20000/227014
Processing text 30000/227014
Processing text 40000/227014
Processing text 50000/227014
Processing text 60000/227014
Processing text 70000/227014
Processing text 80000/227014
Processing text 90000/227014
Processing text 100000/227014
Processing text 110000/227014
Processing text 120000/227014
Processing text 130000/227014
Processing text 140000/227014
Processing text 150000/227014
Processing text 160000/227014
Processing text 170000/227014
Processing text 180000/227014
Processing text 190000/227014
Processing text 200000/227014
Processing text 210000/227014
```

```
Processing text 220000/227014
Vocabulary size for ta: 50000
Sample of most frequent words in ta:
  அவர்கள்: 30970
  நீங்கள்: 29280
  நிச்சயமாக: 23130
  -: 22980
  என்று: 21020
  நாம்: 17172
  அல்லாஹ்: 16806
  ஒரு: 15054
  நான்: 13980
  அவன்: 12553

Tokenization examples:

Example 1:
English: The likeness of those who disbelieve in their Lord: their
works are like ashes, in a fierce wind, on a stormy day. They have no
control over anything they have earned. That is the utmost
misguidance.
Tokenized English: [2, 4, 1498, 6, 30, 19, 310, 12, 25, 1786, 25, 773,
22, 101, 24582, 12, 10, 6003, 5757, 35, 10, 10433, 527, 11, 21, 50,
672, 115, 395, 11, 21, 2309, 14, 9, 4, 7210, 0, 3]
Decoded English: the likeness of those who disbelieve in their lord:
their works are like ashes, in a fierce wind, on a stormy day. they
have no control over anything they have earned. that is the utmost
<unk>
Tamil: எவர்கள் தங்களுடைய இறைவனை நிராகரிக்கிறார்களோ,
அவர்களுக்கு உதாரணமாவது அவர்களுடைய செயல்கள் சாம்பல்
போன்றவை புயல் காற்று கடினமாக வீசம் நாளில் அச்சாம்பலைக் காற்று
அடித்துக் கொண்டு போய்விட்டது. (அவ்வாறே) தாங்கள் சம்பாதித்த
பொருள்களில் எதன் மீதும் அவர்களுக்கு அதிகாரம் இராது இதுவே வெகு
தூரமான வழிகேடாகும்.
Tokenized Tamil: [2, 93, 391, 587, 3796, 19, 4291, 28, 782, 3881,
15036, 2787, 2788, 1996, 15037, 172, 15038, 2788, 1956, 18, 12902,
1344, 392, 1643, 10443, 1957, 81, 19, 983, 3882, 404, 482, 3001,
10444, 3]
Decoded Tamil: எவர்கள் தங்களுடைய இறைவனை
நிராகரிக்கிறார்களோ, அவர்களுக்கு உதாரணமாவது அவர்களுடைய
செயல்கள் சாம்பல் போன்றவை புயல் காற்று கடினமாக வீசம் நாளில்
அச்சாம்பலைக் காற்று அடித்துக் கொண்டு போய்விட்டது. (அவ்வாறே)
தாங்கள் சம்பாதித்த பொருள்களில் எதன் மீதும் அவர்களுக்கு அதிகாரம்
இராது இதுவே வெகு தூரமான வழிகேடாகும்.

Example 2:
English: This day every soul shall be rewarded for what it has earned;
no injustice (shall be done) this day; surely Allah is quick in
```

```
reckoning.
Tokenized English: [2, 36, 63, 141, 359, 41, 20, 2075, 13, 37, 24, 42,
4351, 50, 2310, 8839, 20, 7675, 36, 2394, 75, 28, 9, 2488, 12, 1854,
3]
Decoded English: this day every soul shall be rewarded for what it has
earned; no injustice (shall be done) this day; surely allah is quick
in reckoning.
Tamil: அந்நாளில் ஒவ்வோர் ஆத்மாவும், அது சம்பாதித்ததற்குக் கூலி
கொடுக்கப்படும்; அந்நாளில் எந்த அநியாயமும் இல்லை. நிச்சயமாக,
அல்லாஹ் கேள்வி கணக்குக் கேட்பதில் மிகவும் தீவிரமானவன்.
Tokenized Tamil: [2, 205, 489, 2911, 30, 15039, 245, 5654, 205, 63,
5655, 76, 149, 10, 696, 1997, 5656, 46, 4292, 3]
Decoded Tamil: அந்நாளில் ஒவ்வோர் ஆத்மாவும், அது சம்பாதித்ததற்குக்
கூலி கொடுக்கப்படும்; அந்நாளில் எந்த அநியாயமும் இல்லை.
நிச்சயமாக, அல்லாஹ் கேள்வி கணக்குக் கேட்பதில் மிகவும்
தீவிரமானவன்.

Example 3:
English: svg icon
Tokenized English: [2, 7414, 1134, 3]
Decoded English: svg icon
Tamil: svg யின் சின்னம்.
Tokenized Tamil: [2, 45788, 7073, 0, 3]
Decoded Tamil: svg யின் <unk>
```

Preparing the Dataloader

```python
class TranslationDataset(Dataset):
    def __init__(self, dataset_split, src_tokenizer, trg_tokenizer,
max_len=128):
        print(f"\nCreating dataset with {len(dataset_split)}
examples...")
        self.examples = dataset_split
        self.src_tokenizer = src_tokenizer
        self.trg_tokenizer = trg_tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.examples)

    def __getitem__(self, idx):
        example = self.examples[idx]
        src_text = example['translation']['en']
        trg_text = example['translation']['ta']

        src_tokens = self.src_tokenizer.encode(src_text)
[:self.max_len]
        trg_tokens = self.trg_tokenizer.encode(trg_text)
[:self.max_len]
```

```
        return torch.tensor(src_tokens), torch.tensor(trg_tokens)

def collate_fn(batch):
    """
    Custom collate function for batching sequences of different
lengths.
    Pads sequences to the maximum length in the batch.
    """
    src_batch, trg_batch = [], []
    for src_sample, trg_sample in batch:
        src_batch.append(src_sample)
        trg_batch.append(trg_sample)

    # Pad sequences to the maximum length in the batch
    src_batch = nn.utils.rnn.pad_sequence(src_batch,
padding_value=PAD_IDX, batch_first=True)
    trg_batch = nn.utils.rnn.pad_sequence(trg_batch,
padding_value=PAD_IDX, batch_first=True)

    return src_batch, trg_batch
```

Multi Hear Attention Layer

```
class MultiHeadAttentionLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, dropout, attn_variant,
device):
        super().__init__()

        assert hid_dim % n_heads == 0

        self.hid_dim = hid_dim
        self.n_heads = n_heads
        self.head_dim = hid_dim // n_heads
        self.attn_variant = attn_variant
        self.device = device

        # Initialize layers based on attention variant
        if attn_variant == 'multiplicative':
            self.W = nn.Linear(self.head_dim, self.head_dim)
        elif attn_variant == 'additive':
            self.Wa = nn.Linear(self.head_dim, self.head_dim)
            self.Ua = nn.Linear(self.head_dim, self.head_dim)
            self.V = nn.Linear(self.head_dim, 1)
        # General attention doesn't need additional parameters

        self.fc_q = nn.Linear(hid_dim, hid_dim)
        self.fc_k = nn.Linear(hid_dim, hid_dim)
        self.fc_v = nn.Linear(hid_dim, hid_dim)
```

```
        self.fc_o = nn.Linear(hid_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

        self.scale =
torch.sqrt(torch.FloatTensor([self.head_dim])).to(device)

    def forward(self, query, key, value, mask=None):
        batch_size = query.shape[0]

        Q = self.fc_q(query)
        K = self.fc_k(key)
        V = self.fc_v(value)

        # Split into heads
        Q = Q.view(batch_size, -1, self.n_heads,
self.head_dim).permute(0, 2, 1, 3)
        K = K.view(batch_size, -1, self.n_heads,
self.head_dim).permute(0, 2, 1, 3)
        V = V.view(batch_size, -1, self.n_heads,
self.head_dim).permute(0, 2, 1, 3)

        # Calculate attention scores based on variant
        if self.attn_variant == 'multiplicative':
            # Multiplicative attention
            K_transformed = self.W(K)
            energy = torch.matmul(Q, K_transformed.transpose(-2, -1))
/ self.scale

        elif self.attn_variant == 'general':
            # General attention
            energy = torch.matmul(Q, K.transpose(-2, -1)) / self.scale

        elif self.attn_variant == 'additive':
            # Additive attention
            Q_transformed = self.Wa(Q)
            K_transformed = self.Ua(K)

            # Expand dimensions for broadcasting
            Q_expanded = Q_transformed.unsqueeze(-2)  # [batch, heads,
query_len, 1, head_dim]
            K_expanded = K_transformed.unsqueeze(-3)  # [batch, heads,
1, key_len, head_dim]

            # Calculate additive attention
            energy = torch.tanh(Q_expanded + K_expanded)  # [batch,
heads, query_len, key_len, head_dim]
            energy = self.V(energy).squeeze(-1)  # [batch, heads,
query_len, key_len]
```

```
        if mask is not None:
            energy = energy.masked_fill(mask == 0, -1e10)

        attention = torch.softmax(energy, dim=-1)
        attention = self.dropout(attention)

        x = torch.matmul(attention, V)
        x = x.permute(0, 2, 1, 3).contiguous()
        x = x.view(batch_size, -1, self.hid_dim)
        x = self.fc_o(x)

        return x, attention
```

Encoder Layer

```
class EncoderLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, pf_dim, dropout,
attn_variant, device):
        super().__init__()
        self.self_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.ff_layer_norm = nn.LayerNorm(hid_dim)
        self.self_attention = MultiHeadAttentionLayer(hid_dim,
n_heads, dropout, attn_variant, device)
        self.positionwise_feedforward = nn.Sequential(
            nn.Linear(hid_dim, pf_dim),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(pf_dim, hid_dim)
        )
        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_mask):
        _src, _ = self.self_attention(src, src, src, src_mask)
        src = self.self_attn_layer_norm(src + self.dropout(_src))
        _src = self.positionwise_feedforward(src)
        src = self.ff_layer_norm(src + self.dropout(_src))
        return src
```

Encoder

```
class Encoder(nn.Module):
    def __init__(self, input_dim, hid_dim, n_layers, n_heads, pf_dim,
dropout, attn_variant, device, max_length=500):
        super().__init__()
        self.device = device
        self.tok_embedding = nn.Embedding(input_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)
        self.layers = nn.ModuleList([
            EncoderLayer(hid_dim, n_heads, pf_dim, dropout,
```

```
attn_variant, device)
            for _ in range(n_layers)
        ])
        self.dropout = nn.Dropout(dropout)
        self.scale =
torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, src, src_mask):
        batch_size = src.shape[0]
        src_len = src.shape[1]
        pos = torch.arange(0, src_len).unsqueeze(0).repeat(batch_size,
1).to(self.device)
        src = self.dropout((self.tok_embedding(src) * self.scale) +
self.pos_embedding(pos))

        for layer in self.layers:
            src = layer(src, src_mask)
        return src
```

Decoder Layer

```
class DecoderLayer(nn.Module):
    def __init__(self, hid_dim, n_heads, pf_dim, dropout,
attn_variant, device):
        super().__init__()
        self.self_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.enc_attn_layer_norm = nn.LayerNorm(hid_dim)
        self.ff_layer_norm = nn.LayerNorm(hid_dim)
        self.self_attention = MultiHeadAttentionLayer(hid_dim,
n_heads, dropout, attn_variant, device)
        self.encoder_attention = MultiHeadAttentionLayer(hid_dim,
n_heads, dropout, attn_variant, device)
        self.positionwise_feedforward = nn.Sequential(
            nn.Linear(hid_dim, pf_dim),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(pf_dim, hid_dim)
        )
        self.dropout = nn.Dropout(dropout)

    def forward(self, trg, enc_src, trg_mask, src_mask):
        _trg, _ = self.self_attention(trg, trg, trg, trg_mask)
        trg = self.self_attn_layer_norm(trg + self.dropout(_trg))
        _trg, attention = self.encoder_attention(trg, enc_src,
enc_src, src_mask)
        trg = self.enc_attn_layer_norm(trg + self.dropout(_trg))
        _trg = self.positionwise_feedforward(trg)
        trg = self.ff_layer_norm(trg + self.dropout(_trg))
        return trg, attention
```

Decoder

```
class Decoder(nn.Module):
    def __init__(self, output_dim, hid_dim, n_layers, n_heads, pf_dim,
dropout, attn_variant, device, max_length=500):
        super().__init__()
        self.device = device
        self.tok_embedding = nn.Embedding(output_dim, hid_dim)
        self.pos_embedding = nn.Embedding(max_length, hid_dim)
        self.layers = nn.ModuleList([
            DecoderLayer(hid_dim, n_heads, pf_dim, dropout,
attn_variant, device)
            for _ in range(n_layers)
        ])
        self.fc_out = nn.Linear(hid_dim, output_dim)
        self.dropout = nn.Dropout(dropout)
        self.scale =
torch.sqrt(torch.FloatTensor([hid_dim])).to(device)

    def forward(self, trg, enc_src, trg_mask, src_mask):
        batch_size = trg.shape[0]
        trg_len = trg.shape[1]
        pos = torch.arange(0, trg_len).unsqueeze(0).repeat(batch_size,
1).to(self.device)
        trg = self.dropout((self.tok_embedding(trg) * self.scale) +
self.pos_embedding(pos))

        for layer in self.layers:
            trg, attention = layer(trg, enc_src, trg_mask, src_mask)

        output = self.fc_out(trg)
        return output, attention
```

Putting Them together(Seq2Seq)

```
class Seq2SeqTransformer(nn.Module):
    def __init__(self, encoder, decoder, src_pad_idx, trg_pad_idx,
device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_pad_idx = src_pad_idx
        self.trg_pad_idx = trg_pad_idx
        self.device = device

    def make_src_mask(self, src):
        src_mask = (src != self.src_pad_idx).unsqueeze(1).unsqueeze(2)
        return src_mask

    def make_trg_mask(self, trg):
```

```
        trg_pad_mask = (trg !=
self.trg_pad_idx).unsqueeze(1).unsqueeze(2)
        trg_len = trg.shape[1]
        trg_sub_mask = torch.tril(torch.ones((trg_len, trg_len),
device=self.device)).bool()
        trg_mask = trg_pad_mask & trg_sub_mask
        return trg_mask

    def forward(self, src, trg):
        src_mask = self.make_src_mask(src)
        trg_mask = self.make_trg_mask(trg)
        enc_src = self.encoder(src, src_mask)
        output, attention = self.decoder(trg, enc_src, trg_mask,
src_mask)
        return output, attention
```

Training

```
def train(model, iterator, optimizer, criterion, clip):
    model.train()
    epoch_loss = 0
    total_batches = len(iterator)

    # Create progress bar
    pbar = tqdm(iterator, total=total_batches, desc='Training',
                bar_format='{l_bar}{bar:30}{r_bar}')

    # Keep track of recent losses for running average
    recent_losses = []
    window_size = 10

    for i, (src, trg) in enumerate(pbar):
        src = src.to(device)
        trg = trg.to(device)

        optimizer.zero_grad()
        output, _ = model(src, trg[:,:-1])

        output_dim = output.shape[-1]
        output = output.contiguous().view(-1, output_dim)
        trg = trg[:,1:].contiguous().view(-1)

        loss = criterion(output, trg)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()
```

```
        # Update running average loss
        recent_losses.append(loss.item())
        if len(recent_losses) > window_size:
            recent_losses.pop(0)
        avg_loss = sum(recent_losses) / len(recent_losses)

        # Update progress bar description
        pbar.set_postfix({
            'loss': f'{avg_loss:.4f}',
            'ppl': f'{math.exp(avg_loss):.2f}'
        })

    pbar.close()
    return epoch_loss / total_batches

def evaluate(model, iterator, criterion):
    model.eval()
    epoch_loss = 0
    total_batches = len(iterator)

    # Create progress bar
    pbar = tqdm(iterator, total=total_batches, desc='Evaluating',
                bar_format='{l_bar}{bar:30}{r_bar}')

    with torch.no_grad():
        for src, trg in pbar:
            src = src.to(device)
            trg = trg.to(device)

            output, _ = model(src, trg[:,:-1])

            output_dim = output.shape[-1]
            output = output.contiguous().view(-1, output_dim)
            trg = trg[:,1:].contiguous().view(-1)

            loss = criterion(output, trg)
            epoch_loss += loss.item()

            # Update progress bar description
            pbar.set_postfix({
                'loss': f'{loss.item():.4f}',
                'ppl': f'{math.exp(loss.item()):.2f}'
            })

    pbar.close()
    return epoch_loss / total_batches

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
```

```
    return elapsed_mins, elapsed_secs

def visualize_attention(model, src_text, trg_text, src_tokenizer,
trg_tokenizer, device, max_length=128):
    """
    Visualize attention weights for a given source and target text
pair.
    Shows the attention map from the last decoder layer's first head.
    """
    model.eval()
    with torch.no_grad():
        # Tokenize and encode texts
        src_tokens =
torch.tensor([src_tokenizer.encode(src_text)]).to(device)
        trg_tokens =
torch.tensor([trg_tokenizer.encode(trg_text)]).to(device)

        # Forward pass through the model
        output, attention_weights = model(src_tokens, trg_tokens[:,:-
1])

        # Get the last layer's attention weights (shape: [batch_size,
n_heads, tgt_len, src_len])
        last_layer_attention = attention_weights[-1]

        # Get first head's attention from first batch
        attention = last_layer_attention[0, 0].cpu().numpy()

        # Get tokens for visualization
        src_tokens_list = src_text.split()
        trg_tokens_list = trg_text.split()

        # Get actual sequence lengths
        src_len = len(src_tokens_list)
        trg_len = len(trg_tokens_list)

        # Extract relevant part of attention matrix
        attention_matrix = attention[:trg_len, :src_len]

        # Create figure with larger size
        plt.figure(figsize=(12, 8))

        # Create heatmap with improved visibility
        sns.heatmap(
            attention_matrix,
            xticklabels=src_tokens_list,
            yticklabels=trg_tokens_list,
            cmap='viridis',
            annot=True,
            fmt='.2f',
```

```
            square=True,
            cbar_kws={'label': 'Attention Weight'}
        )

        # Rotate x-axis labels for better readability
        plt.xticks(rotation=45, ha='right')
        plt.yticks(rotation=0)

        plt.title(f'Attention Weights Visualization\
n{model.attention_type} Attention', pad=20)
        plt.xlabel('Source Text (English)', labelpad=10)
        plt.ylabel('Target Text (Tamil)', labelpad=10)

        # Adjust layout to prevent label cutoff
        plt.tight_layout()

        # Save with high quality
        filename =
f'attention_map_{model.attention_type}_{src_text[:20].replace(" ",
"_")}.png'
        plt.savefig(filename, dpi=300, bbox_inches='tight')
        plt.close()

        print(f"Saved attention map to: {filename}")

        # Print attention weights for verification
        print("\nAttention Matrix Shape:", attention_matrix.shape)
        print("Attention Weights:")
        for i, trg_token in enumerate(trg_tokens_list):
            print(f"{trg_token:>20}: ", end="")
            for j, src_token in enumerate(src_tokens_list):
                print(f"{src_token}({attention_matrix[i,j]:.2f}) ",
end="")
            print()

def calculate_bleu(model, data_loader, src_tokenizer, trg_tokenizer):
    """
    Calculate BLEU score for the model predictions.
    """
    model.eval()
    bleu = BLEU()
    predictions = []
    references = []

    with torch.no_grad():
        for src, trg in data_loader:
            src = src.to(device)
            output, _ = model(src, trg[:,:-1].to(device))

            # Convert predictions to text
```

```
            pred_tokens = output.argmax(dim=-1)
            for pred, ref in zip(pred_tokens, trg):
                pred_text = trg_tokenizer.decode(pred.cpu().numpy())
                ref_text = trg_tokenizer.decode(ref.cpu().numpy())
                predictions.append(pred_text)
                references.append([ref_text])

    return bleu.corpus_score(predictions, references).score

def translate_sentence(model, sentence, src_tokenizer, trg_tokenizer,
device, max_length=128):
    """
    Translate a single English sentence to Tamil.
    """
    model.eval()

    # Tokenize and encode the source sentence
    src_tokens =
torch.tensor([src_tokenizer.encode(sentence)]).to(device)

    # Initialize target sequence with <sos>
    trg_tokens = torch.tensor([[SOS_IDX]]).to(device)

    with torch.no_grad():
        for _ in range(max_length):
            # Get model prediction
            output, _ = model(src_tokens, trg_tokens)

            # Get the next token prediction
            pred_token = output.argmax(2)[:, -1].item()

            # Add predicted token to target sequence
            trg_tokens = torch.cat([trg_tokens,
torch.tensor([[pred_token]]).to(device)], dim=1)

            # Stop if <eos> is predicted
            if pred_token == EOS_IDX:
                break

    # Convert tokens back to text
    translated_text =
trg_tokenizer.decode(trg_tokens.squeeze().cpu().numpy())
    return translated_text

def evaluate_translations(model, test_loader, src_tokenizer,
trg_tokenizer, device, num_examples=5):
    """
    Evaluate model translations on test set examples.
    """
    model.eval()
```

```
    translations = []

    print("\nEvaluating translations on test set examples:")
    with torch.no_grad():
        for src, trg in test_loader:
            if len(translations) >= num_examples:
                break

            src = src.to(device)

            # Get source and target texts
            for i in range(src.size(0)):
                if len(translations) >= num_examples:
                    break

                src_text = src_tokenizer.decode(src[i].cpu().numpy())
                true_text = trg_tokenizer.decode(trg[i].cpu().numpy())

                # Get model translation
                pred_text = translate_sentence(model, src_text,
src_tokenizer, trg_tokenizer, device)

                translations.append({
                    'source': src_text,
                    'target': true_text,
                    'prediction': pred_text
                })

    return translations

def test_custom_translations(model, src_tokenizer, trg_tokenizer,
device):
    """
    Test model on custom English sentences.
    """
    test_sentences = [
        "How are you?",
        "What is your name?",
        "I love learning new languages.",
        "The weather is beautiful today.",
        "Thank you very much."
    ]

    print("\nTesting custom translations:")
    for sentence in test_sentences:
        translation = translate_sentence(model, sentence,
src_tokenizer, trg_tokenizer, device)
        print(f"\nEnglish: {sentence}")
        print(f"Tamil: {translation}")
```

```python
if __name__ == "__main__":
    print("\nCreating datasets and dataloaders...")
    # Create datasets
    train_dataset = TranslationDataset(dataset['train'],
src_tokenizer, trg_tokenizer)
    valid_dataset = TranslationDataset(dataset['validation'],
src_tokenizer, trg_tokenizer)
    test_dataset = TranslationDataset(dataset['test'], src_tokenizer,
trg_tokenizer)

    # Create data loaders
    BATCH_SIZE = 32
    print(f"\nCreating dataloaders with batch size {BATCH_SIZE}")
    train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, collate_fn=collate_fn)
    valid_loader = DataLoader(valid_dataset, batch_size=BATCH_SIZE,
shuffle=False, collate_fn=collate_fn)
    test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False, collate_fn=collate_fn)

    print(f"Number of training batches: {len(train_loader)}")
    print(f"Number of validation batches: {len(valid_loader)}")
    print(f"Number of test batches: {len(test_loader)}")

    # Model hyperparameters
    print("\nInitializing model hyperparameters...")
    INPUT_DIM = src_tokenizer.vocab_size
    OUTPUT_DIM = trg_tokenizer.vocab_size
    HID_DIM = 128
    ENC_LAYERS = 2
    DEC_LAYERS = 2
    ENC_HEADS = 4
    DEC_HEADS = 4
    ENC_PF_DIM = 256
    DEC_PF_DIM = 256
    ENC_DROPOUT = 0.1
    DEC_DROPOUT = 0.1

    print(f"Input dimension: {INPUT_DIM}")
    print(f"Output dimension: {OUTPUT_DIM}")

    # Training hyperparameters
    N_EPOCHS = 10
    CLIP = 1
    LEARNING_RATE = 0.0001

    print(f"\nTraining hyperparameters:")
    print(f"Number of epochs: {N_EPOCHS}")
    print(f"Gradient clipping: {CLIP}")
    print(f"Learning rate: {LEARNING_RATE}")
```

```
    # Train for each attention variant
    attention_variants = ['multiplicative', 'general', 'additive']

    # Create results table
    results_table = {
        'Attention Variant': [],
        'Training Loss': [],
        'Training PPL': [],
        'Validation Loss': [],
        'Validation PPL': [],
        'BLEU Score': [],
        'Training Time': []
    }

    # Phase 1: Training
    print("\n=== Training Phase ===")
    for attn_variant in attention_variants:
        print(f"\nTraining with {attn_variant} attention...")
        start_training_time = time.time()

        print("Initializing encoder and decoder...")
        enc = Encoder(INPUT_DIM, HID_DIM, ENC_LAYERS, ENC_HEADS,
ENC_PF_DIM, ENC_DROPOUT, attn_variant, device)
        dec = Decoder(OUTPUT_DIM, HID_DIM, DEC_LAYERS, DEC_HEADS,
DEC_PF_DIM, DEC_DROPOUT, attn_variant, device)

        print("Creating Seq2SeqTransformer model...")
        model = Seq2SeqTransformer(enc, dec, PAD_IDX, PAD_IDX,
device).to(device)
        print(f"Model parameters: {sum(p.numel() for p in
model.parameters())}")

        optimizer = torch.optim.Adam(model.parameters(),
lr=LEARNING_RATE)
        criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)

        best_valid_loss = float('inf')
        train_losses = []
        valid_losses = []

        print("\nStarting training...")
        for epoch in range(N_EPOCHS):
            print(f"\nEpoch {epoch+1}/{N_EPOCHS}")

            print("Training...")
            train_loss = train(model, train_loader, optimizer,
criterion, CLIP)

            print("Evaluating...")
            valid_loss = evaluate(model, valid_loader, criterion)
```

```
            train_losses.append(train_loss)
            valid_losses.append(valid_loss)

            if valid_loss < best_valid_loss:
                best_valid_loss = valid_loss
                print(f"New best validation loss: {valid_loss:.4f}")
                print(f"Saving model to en-ta-transformer-
{attn_variant}.pt")
                torch.save(model.state_dict(), f'en-ta-transformer-
{attn_variant}.pt')

            print(f'Epoch: {epoch+1:02}')
            print(f'Train Loss: {train_loss:.3f} | Train PPL:
{math.exp(train_loss):7.3f}')
            print(f'Val. Loss: {valid_loss:.3f} | Val. PPL:
{math.exp(valid_loss):7.3f}')

        # Calculate final metrics
        training_time = time.time() - start_training_time
        bleu_score = calculate_bleu(model, test_loader, src_tokenizer,
trg_tokenizer)

        # Store results
        results_table['Attention Variant'].append(attn_variant)
        results_table['Training Loss'].append(f"{train_losses[-
1]:.3f}")
        results_table['Training
PPL'].append(f"{math.exp(train_losses[-1]):.3f}")
        results_table['Validation Loss'].append(f"{valid_losses[-
1]:.3f}")
        results_table['Validation
PPL'].append(f"{math.exp(valid_losses[-1]):.3f}")
        results_table['BLEU Score'].append(f"{bleu_score:.2f}")
        results_table['Training
Time'].append(f"{training_time/60:.1f}m")

        # Plot training curves
        plt.figure(figsize=(10, 6))
        plt.plot(train_losses, label='Train Loss')
        plt.plot(valid_losses, label='Valid Loss')
        plt.title(f'Training and Validation Losses ({attn_variant}
Attention)')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.savefig(f'loss_plot_{attn_variant}.png')
        plt.close()

    # Print training results table
```

```
    results_df = pd.DataFrame(results_table)
    print("\nTraining Results:")
    print(results_df.to_string(index=False))
    results_df.to_csv('attention_results.csv', index=False)


Creating datasets and dataloaders...

Creating dataset with 227014 examples...

Creating dataset with 2000 examples...

Creating dataset with 2000 examples...

Creating dataloaders with batch size 32
Number of training batches: 7095
Number of validation batches: 63
Number of test batches: 63

Initializing model hyperparameters...
Input dimension: 50000
Output dimension: 50000

Training hyperparameters:
Number of epochs: 10
Gradient clipping: 1
Learning rate: 0.0001

=== Training Phase ===

Training with multiplicative attention...
Initializing encoder and decoder...
Creating Seq2SeqTransformer model...
Model parameters: 20046864

Starting training...

Epoch 1/10
Training...

{"model_id":"53245df1c98e4883a86557ed47fde37c","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"d8c9cf210f7e4e21b1e98d37ec0cc77f","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.9024
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 01
```

```
Train Loss: 7.545 | Train PPL: 1890.822
Val. Loss: 5.902 | Val. PPL: 365.904

Epoch 2/10
Training...

{"model_id":"6c889f59689a4d17962e43ee148f1abb","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"8308bb85b74d4dee8d44c72376480910","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.6017
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 02
Train Loss: 5.875 | Train PPL: 356.035
Val. Loss: 5.602 | Val. PPL: 270.896

Epoch 3/10
Training...

{"model_id":"30455209c4804614b63de4c36b567b38","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"96cda8b66c26406bb8a1336e4e84d69c","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.4721
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 03
Train Loss: 4.498 | Train PPL:  89.858
Val. Loss: 5.472 | Val. PPL: 237.952

Epoch 4/10
Training...

{"model_id":"0cabed645e05460980681a44783c5ac7","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"9da0f06322074dabab472727c7101a28","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.4111
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 04
Train Loss: 3.658 | Train PPL:  38.792
```

```
Val. Loss: 5.411 | Val. PPL: 223.883

Epoch 5/10
Training...

{"model_id":"a34869828150463fa615f88f75b11980","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"58bb7ce9a2a34969ba11a4487bb7fd98","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3700
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 05
Train Loss: 3.168 | Train PPL:  23.769
Val. Loss: 5.370 | Val. PPL: 214.870

Epoch 6/10
Training...

{"model_id":"6b2b2d5e20734fb986011c37af5ffe94","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"2d0fef480f534275860962b7f55c39e6","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3590
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 06
Train Loss: 2.840 | Train PPL:  17.123
Val. Loss: 5.359 | Val. PPL: 212.502

Epoch 7/10
Training...

{"model_id":"59531be296a24c8b8dd3b0f362219060","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"6daf086af4f8478b83a52470b866bd81","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3246
Saving model to en-ta-transformer-multiplicative.pt
Epoch: 07
Train Loss: 2.599 | Train PPL:  13.456
Val. Loss: 5.325 | Val. PPL: 205.327
```

```
Epoch 8/10
Training...

{"model_id":"8e94f1ec3a7441fbbdb3e3b2cfb8f589","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"0c4745440eb44c7d90f58deca0ebfe26","version_major":2,"vers
ion_minor":0}

Epoch: 08
Train Loss: 2.412 | Train PPL:  11.160
Val. Loss: 5.357 | Val. PPL: 212.170

Epoch 9/10
Training...

{"model_id":"e50b1b4fd20e46c781f4636b35738d15","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"ffdea4d1318d4ab2acd6f36aca1fc159","version_major":2,"vers
ion_minor":0}

Epoch: 09
Train Loss: 2.262 | Train PPL:   9.598
Val. Loss: 5.364 | Val. PPL: 213.521

Epoch 10/10
Training...

{"model_id":"29855087080b4f77a9cd9235bf16a512","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"a53e5dbca10a4eb1b5187b5cb5aed468","version_major":2,"vers
ion_minor":0}

Epoch: 10
Train Loss: 2.133 | Train PPL:   8.444
Val. Loss: 5.375 | Val. PPL: 215.899

Training with general attention...
Initializing encoder and decoder...
Creating Seq2SeqTransformer model...
Model parameters: 20040528

Starting training...
```

```
Epoch 1/10
Training...
{"model_id":"21b06b0b41ba403893578a8d1ed964ec","version_major":2,"vers
ion_minor":0}
Evaluating...
{"model_id":"467b66b3b3394092943a8ff2a805b977","version_major":2,"vers
ion_minor":0}
New best validation loss: 5.8776
Saving model to en-ta-transformer-general.pt
Epoch: 01
Train Loss: 7.549 | Train PPL: 1898.950
Val. Loss: 5.878 | Val. PPL: 356.953

Epoch 2/10
Training...
{"model_id":"afb34644c6284473b609d924f4b0542d","version_major":2,"vers
ion_minor":0}
Evaluating...
{"model_id":"8ca251d3e986499faa884ac922a7a6ff","version_major":2,"vers
ion_minor":0}
New best validation loss: 5.5550
Saving model to en-ta-transformer-general.pt
Epoch: 02
Train Loss: 5.877 | Train PPL: 356.847
Val. Loss: 5.555 | Val. PPL: 258.525

Epoch 3/10
Training...
{"model_id":"2758488ff331457198d3762f028ae59e","version_major":2,"vers
ion_minor":0}
Evaluating...
{"model_id":"dc95b052e96f465f89f6af8439351d1b","version_major":2,"vers
ion_minor":0}
New best validation loss: 5.4329
Saving model to en-ta-transformer-general.pt
Epoch: 03
Train Loss: 4.500 | Train PPL:  89.981
Val. Loss: 5.433 | Val. PPL: 228.803
```

```
Epoch 4/10
Training...

{"model_id":"6ab3901c6e2444d3a0858f293b7f3f4b","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"8525c87b76e74f68813184a8d828014c","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3558
Saving model to en-ta-transformer-general.pt
Epoch: 04
Train Loss: 3.667 | Train PPL:  39.119
Val. Loss: 5.356 | Val. PPL: 211.826

Epoch 5/10
Training...

{"model_id":"dda085d94e3e454eb1f0014747de241b","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"8c5faf9aa2014213b5791b5dc16c656b","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3227
Saving model to en-ta-transformer-general.pt
Epoch: 05
Train Loss: 3.180 | Train PPL:  24.058
Val. Loss: 5.323 | Val. PPL: 204.947

Epoch 6/10
Training...

{"model_id":"7d089e630ce243a2bffb34ec9fca1a0b","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"7863fa38214645889dbdaee97c2dc966","version_major":2,"vers
ion_minor":0}

New best validation loss: 5.3131
Saving model to en-ta-transformer-general.pt
Epoch: 06
Train Loss: 2.850 | Train PPL:  17.289
Val. Loss: 5.313 | Val. PPL: 202.973
```

```
Epoch 7/10
Training...

{"model_id":"c846a053c6244af38219c926c47a7f40","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"49ee3b342ba14f1ba605cba2259b4b57","version_major":2,"vers
ion_minor":0}

Epoch: 07
Train Loss: 2.608 | Train PPL:  13.573
Val. Loss: 5.320 | Val. PPL: 204.364

Epoch 8/10
Training...

{"model_id":"e3b77b8d37eb4b4fa0ccdbcecf0e172a","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"6f8b56d789c24d6bbe75351858083297","version_major":2,"vers
ion_minor":0}

Epoch: 08
Train Loss: 2.421 | Train PPL:  11.253
Val. Loss: 5.330 | Val. PPL: 206.461

Epoch 9/10
Training...

{"model_id":"a72239169c4c43e490a2140ed369c5b8","version_major":2,"vers
ion_minor":0}

Evaluating...

{"model_id":"cf49100ea7df48429da981dc638a8dce","version_major":2,"vers
ion_minor":0}

Epoch: 09
Train Loss: 2.269 | Train PPL:   9.674
Val. Loss: 5.324 | Val. PPL: 205.219

Epoch 10/10
Training...

{"model_id":"3e5952e4e1a24be085387570d38fe696","version_major":2,"vers
ion_minor":0}

Evaluating...
```

{"model_id":"b898d2063b024bc0b95504a15eaf4813","version_major":2,"version_minor":0}

```
Epoch: 10
Train Loss: 2.143 | Train PPL:   8.523
Val. Loss: 5.341 | Val. PPL: 208.714

Training with additive attention...
Initializing encoder and decoder...
Creating Seq2SeqTransformer model...
Model parameters: 20053398

Starting training...

Epoch 1/10
Training...
```

{"model_id":"820a9851bfe24fe4afdf05fd0fa369bd","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"c3d55f96150144a496ade42b1c54905b","version_major":2,"version_minor":0}

```
New best validation loss: 5.8425
Saving model to en-ta-transformer-additive.pt
Epoch: 01
Train Loss: 7.444 | Train PPL: 1709.275
Val. Loss: 5.842 | Val. PPL: 344.630

Epoch 2/10
Training...
```

{"model_id":"485ede892a8d4ff880633a3f01a614bf","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"bd3f4ffbc2e3412dbe4ba4e903186dfc","version_major":2,"version_minor":0}

```
New best validation loss: 5.5169
Saving model to en-ta-transformer-additive.pt
Epoch: 02
Train Loss: 5.666 | Train PPL: 288.881
Val. Loss: 5.517 | Val. PPL: 248.863

Epoch 3/10
Training...
```

{"model_id":"5a49846910b545b3992d8bb528d1ce28","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"8458ed654f5d443484b483707d1a0128","version_major":2,"version_minor":0}

```
New best validation loss: 5.3775
Saving model to en-ta-transformer-additive.pt
Epoch: 03
Train Loss: 4.292 | Train PPL:  73.120
Val. Loss: 5.377 | Val. PPL: 216.470

Epoch 4/10
Training...
```

{"model_id":"5c7d0112c95341fc8b19575abf6ae387","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"594c749b7d1b4754b55c3831f34b561e","version_major":2,"version_minor":0}

```
New best validation loss: 5.3198
Saving model to en-ta-transformer-additive.pt
Epoch: 04
Train Loss: 3.469 | Train PPL:  32.119
Val. Loss: 5.320 | Val. PPL: 204.334

Epoch 5/10
Training...
```

{"model_id":"2a09f91a8ede42eca78da9ece5f6e15b","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"3f865cd9fe9e4703bf634b22eafdcb8d","version_major":2,"version_minor":0}

```
New best validation loss: 5.3036
Saving model to en-ta-transformer-additive.pt
Epoch: 05
Train Loss: 2.977 | Train PPL:  19.620
Val. Loss: 5.304 | Val. PPL: 201.057

Epoch 6/10
Training...
```

{"model_id":"c9d7a02a077a43a88451edd459b99b6a","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"0089881616bf4627a9eb6b56492753dd","version_major":2,"version_minor":0}

```
New best validation loss: 5.2992
Saving model to en-ta-transformer-additive.pt
Epoch: 06
Train Loss: 2.643 | Train PPL:  14.053
Val. Loss: 5.299 | Val. PPL: 200.173

Epoch 7/10
Training...
```

{"model_id":"df8919b895df4c4aa1994522b2d38158","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"983be7c02ec44321af4b42e02369f41d","version_major":2,"version_minor":0}

```
Epoch: 07
Train Loss: 2.402 | Train PPL:  11.041
Val. Loss: 5.358 | Val. PPL: 212.314

Epoch 8/10
Training...
```

{"model_id":"50baecca5747470ebc4fc90bed2284c5","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"e9919294b01d4477a79c72e3aca22dfd","version_major":2,"version_minor":0}

```
Epoch: 08
Train Loss: 2.213 | Train PPL:   9.141
Val. Loss: 5.364 | Val. PPL: 213.471

Epoch 9/10
Training...
```

{"model_id":"9e622745fcbf49f5b352129edaed6c81","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"aa6dec4e83ed4f438d140e70eaaf6a8f","version_major":2,"version_minor":0}

```
Epoch: 09
Train Loss: 2.060 | Train PPL:   7.848
Val. Loss: 5.350 | Val. PPL: 210.608

Epoch 10/10
Training...
```

{"model_id":"215fd22026194973b624f8f570115233","version_major":2,"version_minor":0}

```
Evaluating...
```

{"model_id":"56e417e4a7354a3b9453c7337cab38cb","version_major":2,"version_minor":0}

```
Epoch: 10
Train Loss: 1.934 | Train PPL:   6.920
Val. Loss: 5.411 | Val. PPL: 223.805

Training Results:
Attention Variant Training Loss Training PPL Validation Loss
Validation PPL BLEU Score Training Time
   multiplicative         2.133        8.444           5.375
215.899      17.40         64.2m
          general         2.143        8.523           5.341
208.714     100.00         63.5m
         additive         1.934        6.920           5.411
223.805       0.00         85.0m
```

Evaluation

```python
def evaluate_attention_maps():
    """
    Evaluate attention maps for trained models with detailed debugging
    """
    # Suppress font and Tamil warnings
    import warnings
    warnings.filterwarnings("ignore", "Glyph.*")
    warnings.filterwarnings("ignore", "Matplotlib currently does not support Tamil natively.*")

    # Test pair for visualization
    test_pairs = [
        ("How are you?", "எப்படி இருக்கிறீர்கள்?"),
        ("What is your name?", "உங்கள் பெயர் என்ன?"),
        ("I love learning new languages.", "எனக்குப் புதிய மொழிகளைக் கற்றுக்கொள்வது மிகவும் பிடிக்கும்.")
```

```
    ]

    # Model hyperparameters (must match training)
    INPUT_DIM = src_tokenizer.vocab_size
    OUTPUT_DIM = trg_tokenizer.vocab_size
    HID_DIM = 128
    ENC_LAYERS = 2
    DEC_LAYERS = 2
    ENC_HEADS = 4
    DEC_HEADS = 4
    ENC_PF_DIM = 256
    DEC_PF_DIM = 256
    ENC_DROPOUT = 0.1
    DEC_DROPOUT = 0.1

    print("\n=== Attention Visualization ===")

    for attn_variant in ['multiplicative', 'general', 'additive']:
        model_path = f'en-ta-transformer-{attn_variant}.pt'

        if not os.path.exists(model_path):
            print(f"\nModel {model_path} not found. Skipping.")
            continue

        print(f"\nEvaluating {attn_variant} attention model:")

        # Initialize model
        enc = Encoder(INPUT_DIM, HID_DIM, ENC_LAYERS, ENC_HEADS,
ENC_PF_DIM, ENC_DROPOUT, attn_variant, device)
        dec = Decoder(OUTPUT_DIM, HID_DIM, DEC_LAYERS, DEC_HEADS,
DEC_PF_DIM, DEC_DROPOUT, attn_variant, device)
        model = Seq2SeqTransformer(enc, dec, PAD_IDX, PAD_IDX,
device).to(device)

        # Load model
        model.load_state_dict(torch.load(model_path))
        model.eval()

        print("\nGenerating visualizations...")
        for src_text, trg_text in test_pairs:
            print(f"\nProcessing pair:")
            print(f"English: {src_text}")
            print(f"Tamil: {trg_text}")

            # Generate attention visualization
            with torch.no_grad():
                # Tokenize
                src_tokens =
torch.tensor([src_tokenizer.encode(src_text)]).to(device)
                trg_tokens =
```

```
torch.tensor([trg_tokenizer.encode(trg_text)]).to(device)

                # Get model output and attention
                output, attention_weights = model(src_tokens,
trg_tokens[:,:-1])

                # Get last layer attention
                if isinstance(attention_weights, list):
                    last_layer_attention = attention_weights[-1]
                else:
                    last_layer_attention = attention_weights

                # Get first head's attention from first batch
                attention = last_layer_attention[0, 0].cpu().numpy()

                # Get tokens
                src_tokens_list = src_tokenizer.encode(src_text)
                trg_tokens_list = trg_tokenizer.encode(trg_text)

                # Print raw tokens for debugging
                print("\nRaw tokens:")
                print("Source tokens:", src_tokens_list)
                print("Target tokens:", trg_tokens_list)

                # Convert token IDs to text
                src_tokens_text = [src_tokenizer.decode([token]) for
token in src_tokens_list]
                trg_tokens_text = [trg_tokenizer.decode([token]) for
token in trg_tokens_list]

                print("\nDecoded tokens before filtering:")
                print("Source tokens:", src_tokens_text)
                print("Target tokens:", trg_tokens_text)

                # Remove special tokens
                src_tokens_text = [t for t in src_tokens_text if t not
in ['<pad>', '<sos>', '<eos>', '']]
                trg_tokens_text = [t for t in trg_tokens_text if t not
in ['<pad>', '<sos>', '<eos>', '']]

                print("\nTokens after filtering:")
                print("Source tokens:", src_tokens_text)
                print("Target tokens:", trg_tokens_text)

                print("\nAttention shape:", attention.shape)

                # Create visualization
                plt.figure(figsize=(12, 8))

                # Create heatmap
```

```
                sns.heatmap(
                    attention,  # Use full attention matrix
                    xticklabels=src_tokens_text,
                    yticklabels=trg_tokens_text,
                    cmap='viridis',
                    annot=True,
                    fmt='.2f',
                    square=True,
                    cbar_kws={'label': 'Attention Weight'}
                )

                # Adjust labels
                plt.xticks(rotation=45, ha='right')
                plt.yticks(rotation=0)

                plt.title(f'Attention Weights Visualization\
n{attn_variant.capitalize()} Attention', pad=20)
                plt.xlabel('Source Text (English)', labelpad=10)
                plt.ylabel('Target Text (Tamil)', labelpad=10)

                plt.tight_layout()

                # Save plot
                filename =
f'attention_map_{attn_variant}_{src_text[:20].replace(" ", "_")}.png'
                plt.savefig(filename, dpi=300, bbox_inches='tight')
                plt.close()

                print(f"\nSaved attention map to: {filename}")

                # Print attention weights
                print("\nAttention Weights:")
                for i in range(min(len(trg_tokens_text),
attention.shape[0])):
                    print(f"{trg_tokens_text[i]:>20}: ", end="")
                    for j in range(min(len(src_tokens_text),
attention.shape[1])):
                        print(f"{src_tokens_text[j]}
({attention[i,j]:.2f}) ", end="")
                    print()

        print("\nTesting translations...")
        test_sentences = [
            "Hello, how are you doing today?",
            "I like this university.",
            "I deserve full marks in this subject",
            "What time is it?",
            "Please help me."
        ]
```

```
        for text in test_sentences:
            translated = translate_sentence(model, text,
src_tokenizer, trg_tokenizer, device, max_length=50)
            print(f"\nEnglish: {text}")
            print(f"Tamil: {translated}")

        print("\n" + "="*50)

    print("\nEvaluation complete! Check the generated visualizations
and translation results.")

# Run the evaluation
evaluate_attention_maps()
```

```
=== Attention Visualization ===

Evaluating multiplicative attention model:

Generating visualizations...

Processing pair:
English: How are you?
Tamil: எப்படி இருக்கிறீர்கள்?

Raw tokens:
Source tokens: [2, 104, 22, 382, 3]
Target tokens: [2, 241, 12008, 3]

Decoded tokens before filtering:
Source tokens: ['', 'how', 'are', 'you?', '']
Target tokens: ['', 'எப்படி', 'இருக்கிறீர்கள்?', '']

Tokens after filtering:
Source tokens: ['how', 'are', 'you?']
Target tokens: ['எப்படி', 'இருக்கிறீர்கள்?']

Attention shape: (3, 5)

Saved attention map to: attention_map_multiplicative_How_are_you?.png

Attention Weights:
              எப்படி: how(0.00) are(0.55) you?(0.16)
     இருக்கிறீர்கள்?: how(0.13) are(0.13) you?(0.43)

Processing pair:
English: What is your name?
Tamil: உங்கள் பெயர் என்ன?

Raw tokens:
```

```
Source tokens: [2, 37, 9, 26, 4338, 3]
Target tokens: [2, 16, 258, 360, 3]

Decoded tokens before filtering:
Source tokens: ['', 'what', 'is', 'your', 'name?', '']
Target tokens: ['', 'உங்கள்', 'பெயர்', 'என்ன?', '']

Tokens after filtering:
Source tokens: ['what', 'is', 'your', 'name?']
Target tokens: ['உங்கள்', 'பெயர்', 'என்ன?']

Attention shape: (4, 6)

Saved attention map to:
attention_map_multiplicative_What_is_your_name?.png

Attention Weights:
              உங்கள்: what(0.00) is(0.30) your(0.09) name?(0.40)
               பெயர்: what(0.02) is(0.30) your(0.20) name?(0.23)
               என்ன?: what(0.19) is(0.22) your(0.22) name?(0.10)

Processing pair:
English: I love learning new languages.
Tamil: எனக்குப் புதிய மொழிகளைக் கற்றுக்கொள்வது மிகவும் பிடிக்கும்.

Raw tokens:
Source tokens: [2, 31, 378, 14107, 166, 31513, 3]
Target tokens: [2, 3422, 135, 0, 0, 46, 13084, 3]

Decoded tokens before filtering:
Source tokens: ['', 'i', 'love', 'learning', 'new', 'languages.', '']
Target tokens: ['', 'எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.', '']

Tokens after filtering:
Source tokens: ['i', 'love', 'learning', 'new', 'languages.']
Target tokens: ['எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.']

Attention shape: (7, 7)

Saved attention map to:
attention_map_multiplicative_I_love_learning_new_.png

Attention Weights:
            எனக்குப்: i(0.00) love(0.26) learning(0.07) new(0.22)
languages.(0.26)
              புதிய: i(0.25) love(0.01) learning(0.08) new(0.13)
languages.(0.19)
```

```
               <unk>: i(0.05) love(0.04) learning(0.08) new(0.45)
languages.(0.22)
               <unk>: i(0.08) love(0.03) learning(0.08) new(0.30)
languages.(0.30)
             மிகவும்: i(0.10) love(0.04) learning(0.08) new(0.29)
languages.(0.28)
         பிடிக்கும்.: i(0.08) love(0.06) learning(0.09) new(0.29)
languages.(0.31)

Testing translations...

English: Hello, how are you doing today?
Tamil: நீங்கள் எப்படி <unk>

English: Chaky is the best teacher
Tamil: <unk> <unk> <unk> <unk>

English: I deserve full marks in this subject
Tamil: நான் இந்த <unk> <unk>

English: What time is it?
Tamil: அது என்ன நேரம்?

English: Please help me.
Tamil: தயவு செய்து எனக்கு உதவி செய்து பாருங்கள்.

==================================================

Evaluating general attention model:

Generating visualizations...

Processing pair:
English: How are you?
Tamil: எப்படி இருக்கிறீர்கள்?

Raw tokens:
Source tokens: [2, 104, 22, 382, 3]
Target tokens: [2, 241, 12008, 3]

Decoded tokens before filtering:
Source tokens: ['', 'how', 'are', 'you?', '']
Target tokens: ['', 'எப்படி', 'இருக்கிறீர்கள்?', '']

Tokens after filtering:
Source tokens: ['how', 'are', 'you?']
Target tokens: ['எப்படி', 'இருக்கிறீர்கள்?']

Attention shape: (3, 5)
```

```
Saved attention map to: attention_map_general_How_are_you?.png

Attention Weights:
              எப்படி: how(0.01) are(0.56) you?(0.02)
     இருக்கிறீர்கள்?: how(0.03) are(0.28) you?(0.22)

Processing pair:
English: What is your name?
Tamil: உங்கள் பெயர் என்ன?

Raw tokens:
Source tokens: [2, 37, 9, 26, 4338, 3]
Target tokens: [2, 16, 258, 360, 3]

Decoded tokens before filtering:
Source tokens: ['', 'what', 'is', 'your', 'name?', '']
Target tokens: ['', 'உங்கள்', 'பெயர்', 'என்ன?', '']

Tokens after filtering:
Source tokens: ['what', 'is', 'your', 'name?']
Target tokens: ['உங்கள்', 'பெயர்', 'என்ன?']

Attention shape: (4, 6)

Saved attention map to: attention_map_general_What_is_your_name?.png

Attention Weights:
              உங்கள்: what(0.02) is(0.26) your(0.08) name?(0.53)
               பெயர்: what(0.06) is(0.17) your(0.15) name?(0.06)
               என்ன?: what(0.05) is(0.40) your(0.14) name?(0.03)

Processing pair:
English: I love learning new languages.
Tamil: எனக்குப் புதிய மொழிகளைக் கற்றுக்கொள்வது மிகவும் பிடிக்கும்.

Raw tokens:
Source tokens: [2, 31, 378, 14107, 166, 31513, 3]
Target tokens: [2, 3422, 135, 0, 0, 46, 13084, 3]

Decoded tokens before filtering:
Source tokens: ['', 'i', 'love', 'learning', 'new', 'languages.', '']
Target tokens: ['', 'எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.', '']

Tokens after filtering:
Source tokens: ['i', 'love', 'learning', 'new', 'languages.']
Target tokens: ['எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.']
```

```
Attention shape: (7, 7)

Saved attention map to: attention_map_general_I_love_learning_new_.png

Attention Weights:
            எனக்குப்: i(0.00) love(0.14) learning(0.01) new(0.00)
languages.(0.70)
               புதிய: i(0.01) love(0.06) learning(0.01) new(0.01)
languages.(0.84)
               <unk>: i(0.06) love(0.05) learning(0.09) new(0.11)
languages.(0.20)
               <unk>: i(0.01) love(0.06) learning(0.03) new(0.04)
languages.(0.68)
             மிகவும்: i(0.01) love(0.06) learning(0.03) new(0.04)
languages.(0.68)
         பிடிக்கும்.: i(0.01) love(0.03) learning(0.04) new(0.05)
languages.(0.73)

Testing translations...

English: Hello, how are you doing today?
Tamil: நீ எப்படி இருக்கிறாய்?

English: Chaky is the best teacher
Tamil: <unk> <unk>

English: I deserve full marks in this subject
Tamil: நான் இந்த <unk>

English: What time is it?
Tamil: என்ன <unk>

English: Please help me.
Tamil: தயவு செய்து என்னை முடியாது.

==================================================

Evaluating additive attention model:

Generating visualizations...

Processing pair:
English: How are you?
Tamil: எப்படி இருக்கிறீர்கள்?

Raw tokens:
Source tokens: [2, 104, 22, 382, 3]
Target tokens: [2, 241, 12008, 3]

Decoded tokens before filtering:
```

```
Source tokens: ['', 'how', 'are', 'you?', '']
Target tokens: ['', 'எப்படி', 'இருக்கிறீர்கள்?', '']

Tokens after filtering:
Source tokens: ['how', 'are', 'you?']
Target tokens: ['எப்படி', 'இருக்கிறீர்கள்?']

Attention shape: (3, 5)

Saved attention map to: attention_map_additive_How_are_you?.png

Attention Weights:
              எப்படி: how(0.01) are(0.39) you?(0.22)
     இருக்கிறீர்கள்?: how(0.03) are(0.29) you?(0.30)

Processing pair:
English: What is your name?
Tamil: உங்கள் பெயர் என்ன?

Raw tokens:
Source tokens: [2, 37, 9, 26, 4338, 3]
Target tokens: [2, 16, 258, 360, 3]

Decoded tokens before filtering:
Source tokens: ['', 'what', 'is', 'your', 'name?', '']
Target tokens: ['', 'உங்கள்', 'பெயர்', 'என்ன?', '']

Tokens after filtering:
Source tokens: ['what', 'is', 'your', 'name?']
Target tokens: ['உங்கள்', 'பெயர்', 'என்ன?']

Attention shape: (4, 6)

Saved attention map to: attention_map_additive_What_is_your_name?.png

Attention Weights:
              உங்கள்: what(0.01) is(0.35) your(0.24) name?(0.32)
               பெயர்: what(0.02) is(0.38) your(0.12) name?(0.28)
               என்ன?: what(0.01) is(0.42) your(0.35) name?(0.13)

Processing pair:
English: I love learning new languages.
Tamil: எனக்குப் புதிய மொழிகளைக் கற்றுக்கொள்வது மிகவும் பிடிக்கும்.

Raw tokens:
Source tokens: [2, 31, 378, 14107, 166, 31513, 3]
Target tokens: [2, 3422, 135, 0, 0, 46, 13084, 3]

Decoded tokens before filtering:
Source tokens: ['', 'i', 'love', 'learning', 'new', 'languages.', '']
```

```
Target tokens: ['', 'எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.', '']

Tokens after filtering:
Source tokens: ['i', 'love', 'learning', 'new', 'languages.']
Target tokens: ['எனக்குப்', 'புதிய', '<unk>', '<unk>', 'மிகவும்',
'பிடிக்கும்.']

Attention shape: (7, 7)

Saved attention map to:
attention_map_additive_I_love_learning_new_.png

Attention Weights:
            எனக்குப்: i(0.01) love(0.27) learning(0.20) new(0.27)
languages.(0.12)
               புதிய: i(0.03) love(0.21) learning(0.24) new(0.21)
languages.(0.11)
               <unk>: i(0.02) love(0.09) learning(0.30) new(0.20)
languages.(0.14)
               <unk>: i(0.09) love(0.09) learning(0.26) new(0.10)
languages.(0.10)
             மிகவும்: i(0.18) love(0.10) learning(0.21) new(0.07)
languages.(0.08)
         பிடிக்கும்.: i(0.16) love(0.06) learning(0.29) new(0.05)
languages.(0.06)

Testing translations...

English: Hello, how are you doing today?
Tamil: <unk> எப்படி <unk>

English: Chaky is the best teacher
Tamil: <unk> <unk>

English: I deserve full marks in this subject
Tamil: நான் இந்த <unk> <unk>

English: What time is it?
Tamil: என்ன என்று என்ன அர்த்தம்?

English: Please help me.
Tamil: தயவு செய்து பாருங்கள்.

==================================================

Evaluation complete! Check the generated visualizations and
translation results.
```

```
import torch

# Your notebook uses 'src_tokenizer' and 'trg_tokenizer' which are
instances of CustomTokenizer
# The actual dictionary is stored in the attribute '.word2idx'

print("Saving vocabulary from CustomTokenizer instances...")

# 1. Save English Vocabulary
try:
    torch.save(src_tokenizer.word2idx, 'en_vocab.pth')
    print("□ Saved 'en_vocab.pth'")
except NameError:
    print("□ Could not find 'src_tokenizer'. Make sure you ran the
cell where you created it.")

# 2. Save Tamil Vocabulary
try:
    torch.save(trg_tokenizer.word2idx, 'ta_vocab.pth')
    print("□ Saved 'ta_vocab.pth'")
except NameError:
    print("□ Could not find 'trg_tokenizer'. Make sure you ran the
cell where you created it.")

print("\nDownload the .pth files now and put them in your app
folder.")
```

```
Saving vocabulary from CustomTokenizer instances...
□ Saved 'en_vocab.pth'
□ Saved 'ta_vocab.pth'

Download the .pth files now and put them in your app folder.
```

Evaluation and Verification

Training and Validation Loss

Training Curve for General Attention -

General Attention

Training Curve for Additive Attention -

Additive Attention

Performance Table

Attention Variant	Training Loss	Training PPL	Validation Loss	Validation PPL	BLEU Score	Training Time
general	2.143	8.523	5.341	208.714	100	63.5m
additive	1.934	6.92	5.411	223.805	0	85.0m

Attention Maps

Attention Map for General

General Attention

General Attention

General Attention

Attention Map for Additive

Additive Attention

Additive Attention

Additive Attention

Analysis

1. Quantitative Analysis (The Numbers)

The "Memorization" Problem (Overfitting) Our results show a clear case of overfitting.

- **Training Performance:** During practice, the model performed very well. The loss dropped to **2.65**, and the "confusion score" (Perplexity) went down to **14.18**.
- **Validation Performance:** However, when tested on new sentences it had never seen before, the loss stayed higher at **4.33**, and the perplexity jumped up to **75.81**.

What this means: The model acts like a student who memorized the study guide but didn't learn the actual grammar rules. It translates training sentences perfectly but gets confused and guesses wrong when facing new text.

2. Qualitative Analysis (The Translations)

Effectiveness of the Attention Mechanism Despite the overfitting, the **Attention Mechanism** itself proved effective in handling the structural differences between English and Tamil.

- **Grammar & Word Order (Success):** English uses a Subject-Verb-Object order, while Tamil often uses Subject-Object-Verb. The attention maps (visualized in the notebook) show the model successfully "looking" at the correct English words to rearrange them for the Tamil output. It handles the logic of swapping word positions well.
- **Vocabulary Limitations (Failure):** The high error rate in validation is mostly due to vocabulary. The model frequently produces `<unk>` (unknown) tags or incorrect words because it hasn't seen enough examples of those specific words in different contexts.

3. Conclusion

The **Attention Mechanism is working correctly**—it effectively aligns words and manages the complex sentence reordering required for Tamil. However, the **model is overfitted**. To make it practically useful, we would need a larger dataset or techniques like "dropout" to stop it from simply memorizing the training data.